03

Creando mi propia excepción

Transcripción

[00:00] Ya hemos visto en el video anterior que las excepciones van extendiendo de distintos niveles de jerarquía entre objetos. Dando una una visión, digamos un poco más gráfica, tenemos este diagrama aquí. Comenzamos desde abajo. Tenemos ArithmeticException y NullPointerException, que son las dos excepciones con las que hemos venido trabajando hasta el momento.

[00:26] Las dos se extiendes de RuntimeException, lo hemos visto en el código. Y RuntimeException extiende a la vez de Exception, el cual a la vez extiende de throwable. Esta es la relación, digamos, jerárquica, que tienen ahora.

[00:43] Entonces, ahora la pregunta es, si yo quiero crear mi propia excepción, si ArithmeticException y NullPointerException extienden the RuntimeException, ¿será que si yo creo una excepción que extienda de RantimeException va a funcionar? Vamos a verlo, vamos al código. Entonces, cerramos todo esto que hemos abierto aquí para explorar la excepción.

[01:09] Y vamos a crear aquí a new class y vamos a llamar MiException, solamente para fines ilustrativos. Perfecto, le doy finish. Aquí yo ya he creado mi clase para mi excepción. ¿Qué es lo que me falta? Extenderla de su clase padre que va a ser RuntimeException. Entonces le voy a dar extends Runtime, "Ctrl + espacio" y la segunda opción de abajo RuntimeException.

[01:46] Listo. Y ahora, bueno, voy a crearle un constructor, porque si no le creo un constructor, él por defecto va a ir a la clase del padre, y yo no quiero eso p

el momento. Voy a crearle aquí un public MiException. Ese un constructor por defecto. ¿Y aquí a quién invoco? Invoco a super, para llamar al constructor de la clase padre.

[02:21] ¿Recuerdan que así estaba de igual forma en la anterior clase ArithmeticException? Y solamente para ahorrar código voy a copiar, voy a pegar y aquí le voy a poner un String message. Y este message lo voy a mandar en super como parámetro y todo bien, va a compilar porque la clase padre también recibe ese parámetro. Perfecto. Entonces ya tengo mi excepción.

[02:44] Regresa a mi clase flujo y ahora voy a borrar esta cuenta que ya no me sirve por ahora, y en vez de ArithmeticException voy a lanzar mi excepción. Está aquí, vieron, incluso el id ya me autocompletó, MiException. Y le voy a mandar aquí un mensaje: "Mi excepcion fue lanzada", perfecto. Y si ejecuto este método, voy a poner "Ctrl + S", ejecuto, guardo. Y listo, ¿aquí qué salió?

[03:22] Main, método 1, método 2 y exception, en el hilo main Mi excepción fue lanzada. Perfecto. ¿Entonces, qué significa? Yo ya he creado mi primera bomba, y lo mejor de todo, la hice explotar. Si yo quiero tratar esta bomba, ¿cómo debería hacer? Voy a venir aquí a método 1, y repasando lo que vimos en la clase anterior, lo que voy a hacer es un try de método 2.

[03:54] Perfecto, porque le voy a decir: "intente ejecutar método 2 para que englobe todo este código". Y, obviamente, un try no puede vivir sin su catch. Entonces le pongo catch. Los paréntesis siempre. Y en catch, ¿será que si le pongo ArithmeticException, NullPointerException, lo va a atrapar? No lo va a atrapar. ¿Por qué?

[04:19] Porque simplemente no es el tipo de objeto que yo estoy referenciando aquí, que es MiException. Por ejemplo, si yo le pongo aquí NullPointerException y le hago aquí printStackTrace, si yo ejecuto este código, nuevamente le digo que sí quiero guardar, él igual va a lanzar MiException.

¿Por qué? Porque él no está preparado para ese tipo de excepción, eso ya lo vimos en la clase anterior.

[04:53] Él no está preparado para lidiar con MiException. Está preparado para un NullPointerException. ¿Qué hago entonces? Muy fácil, le voy a decir entonces "atrapa MiException". MiException me, vamos a cambiarle la variable, el nombre de la variable. Listo, vamos a acomodar esto un poquito aquí, perfecto.

[05:24] Ahora si ejecuto, le doy que sí quiero guardar, y vemos ahí bien claro que él dio un Inicio en main, Inicio método 1, método 2. MiException fue lanzada nuevamente inicio método 2. ¿Por qué? Porque aquí se se nos escapó, yo no borré este método de aquí y lo curioso es claro, él nuevamente llega, entra a método 2 y lanza MiException y aquí acaba el flujo.

[05:50] Eso es porque yo estoy ejecutando dos veces el método 2, entonces ese fue un error mío. Yo no quiero esto, obviamente. Guardo aquí y nuevamente intento, y ahora si el juegos está como esperábamos. ¿Por qué? Porque él dio Inicio main, Inicio método 1, Inicio método 2, lanzó MiException. La exception que acabamos de crear nosotros en este mismo momento. ¿Y qué hizo?

[06:15] Trató la excepción, la recibió, imprimió el StackTrace, el la traza de la pila de ejecución y dijo: "perfecto, ahora voy a seguir con mi vida," sigue todo, Fin de método 1 y Fin de main. Y vemos que, claro, obviamente, después de throw new MiException no imprime nada, porque ya vimos que no puedes escribir nada más después del throw. Ahora llega una nueva duda.

[06:46] ¿Por qué entonces yo estoy usando MiException, extendiendo de RuntimeException y no estoy extendiendo directamente de Excepción o, mejor aún, de throwable? Porque ya vimos que es throwablel quien tiene toda la lógica, todos los métodos que usa exceptión, porque ArithmeticException, RuntimeException, Exception todos lo único que hacen ¿qué es? Es llamar super.

[07:17] Super, super, super, para llamar a la clase padre siempre. ¿Y al final quién hace todo? Throwable. Entonces, eso es un poco sospechoso. Vamos a ver en el siguiente video un poco más acerca de cómo están organizadas las excepciones en Java.